Distributed Systems

Introduction

What is distributed system?

- Everywhere in nature we find examples of distribution of activities ???? Give examples!!!!
- Thus, centralizing of activities is not the natural way of handling things.
- Centralization is the opposite of distribution.
- Make definition of centralization !!!!
- Parallel versus sequential.
- Time is the key issue in parallel versus sequential comparison.
- If we can process certain tasks at the same time, we are processing in parallel.

Motivation towards Distributed systems

- Many applications are inherently distributed like banks, universities, ….
- Sharing of resources (can be HW like CPU, printers, memory or can be SW like data, database, programs)

- <u>Definition of Distributed system</u>
- There are many definitions of distributed system we will concentrate on some of them.
- ***1- A distributed system is a collection of independent computers that appear to the users of the system as a single computer***

Definition of Distributed system

- 2- A group of connected components that are cooperating to perform a single task (in parallel).
- 3- Interconnected devices that are sharing data and resources.

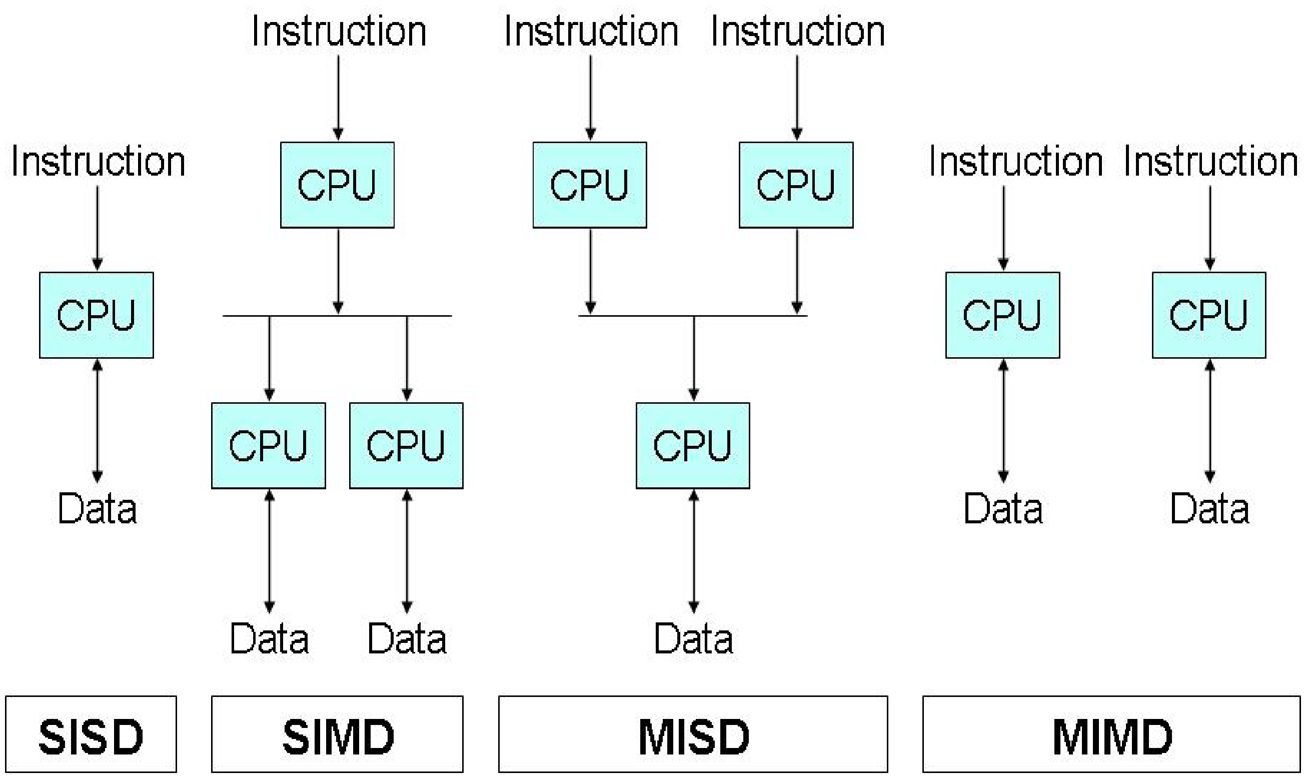
- All these definitions share the presence of more than component or device that are doing something together.

Flynn's taxonomy for computer's HW

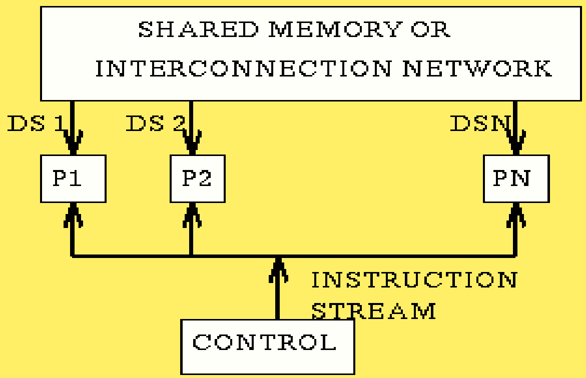
Computer HW

- SISD
  Single Instruction Single Data
  (Ordinary PC)
- SIMD
  Single Instruction Multiple Data
  (Array processor)
- MISD
  Multiple Instruction Single Data
  (Pipelined processors)
- MIMD
  Multiple **Instruction** Multiple **Data**
  (Multiprocessors or multicomputers)

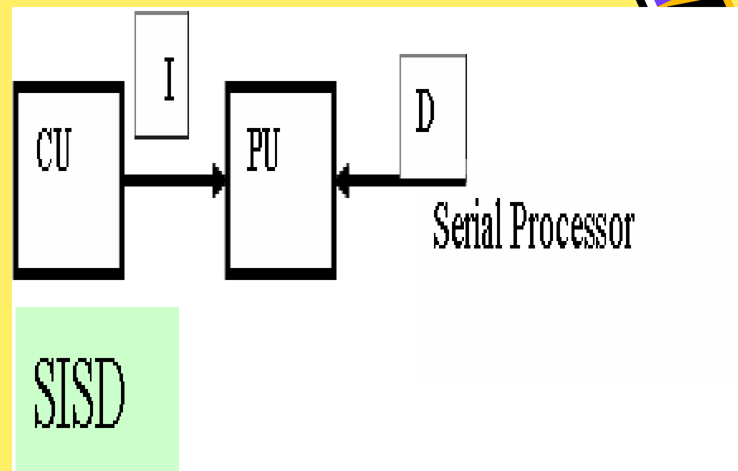
Flynn's Taxonomy

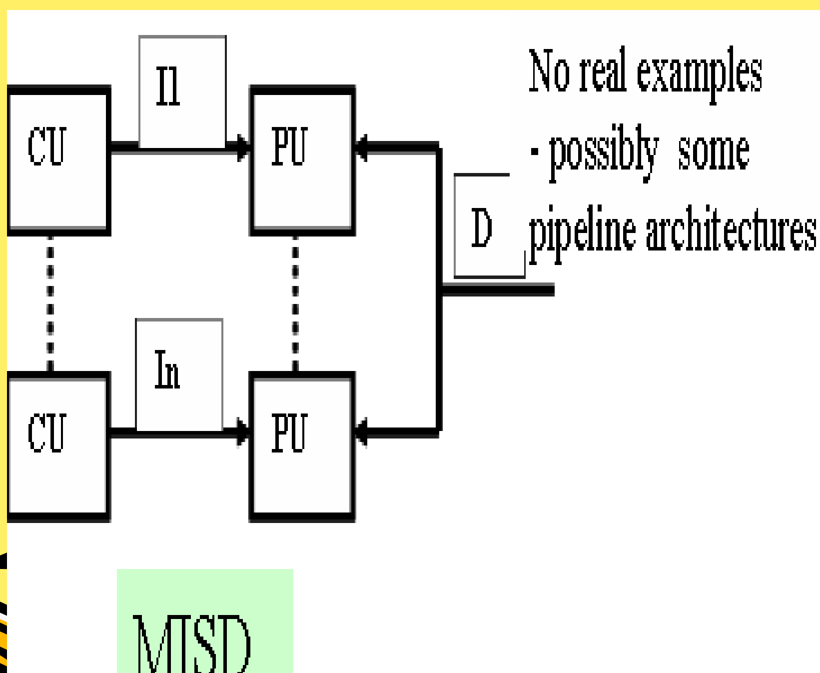


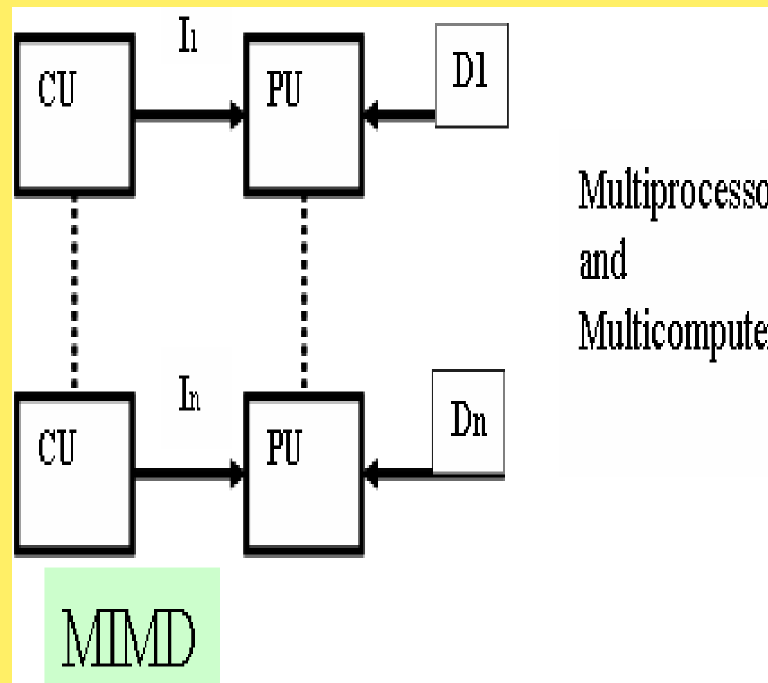
SIMD



P = PROCESSOR
DS = DATA STREAM

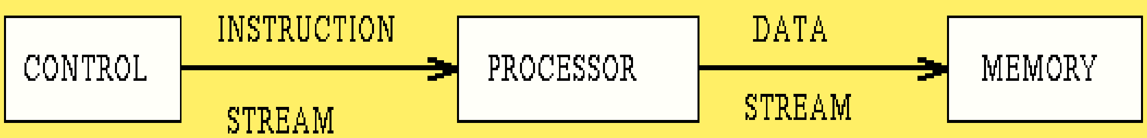






(SISD) single instruction stream and a single data stream

- It has only one CPU and also called von Neumann computers, that starts from personal computers (PC) to large mainframes.
- A sequential machine process one instruction at a time on single data in single processing element.
- Communication are made between memory and CPU using Bus.
- Speed limited by CPU speed and bus speed.



(SISD) single instruction stream and a single data stream

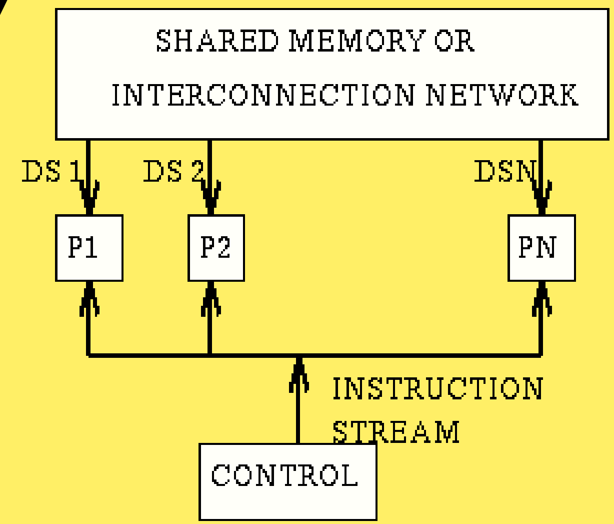
- To compute the sum of N numbers a_1, a_2, ... a_N the processor needs to gain access to memory N consecutive times (to receive one number).
- N-1 additions are executed in sequence. Therefore the computation takes O(N) operations. i.e. algorithms for SISD computers do not contain any parallelism, there is only one processor
- <u>To speed up</u>, add concurrency by:

1- Concurrent execution of several different user's programs (multiprogramming). More than program share the CPU: one program uses CPU for some time and when it releases the CPU another program can use it (for example, round robin time scheduling technique).

- 2- Execute I/O operations simultaneously with the execution of user's program (do you know what is this called???).
- 3- Using parallel functional units inside the processing unit to be multifunction PU. Each unit will be assigned a specific function and has its own data stream so as more than one unit can function at the same time.
- 4- pipelining where the instruction or operation is broken into its element parts and each part is assigned to a processing element. Thus, phases of the instruction are now pipelined including fetching, decodng, operand fetch, ALU execution.

SIMD
Single Instruction Multiple Data
(Array and vector processor)

- All N identical processors operate under the control of a single instruction stream issued by a central control unit.
- There are N data streams, one per processor so different data can be used in each processor.
- The processors operate **synchronously** and a **global clock** is used to ensure **lockstep** operation. i.e. at each step (global clock tick) all processors execute the same instruction, each on a different data stream.



P = PROCESSOR
DS = DATA STREAM

SIMD
Single Instruction Multiple Data
(Array and vector processor)

- It is best used in applications that contains operation on matrices or vectors.
- Each processor will execute the same operation but on different data set which is usually a part of the matrix.
- The total result will be stored in the shared memory.
- The processing elements (PEs) are slaves under control of a master CU.
- The CU has many functions like: decoding the instruction, generating micro-operations needed for execution for the PEs, generating and broadcast common memory addresses and receiving the data from PEs and producing the final results

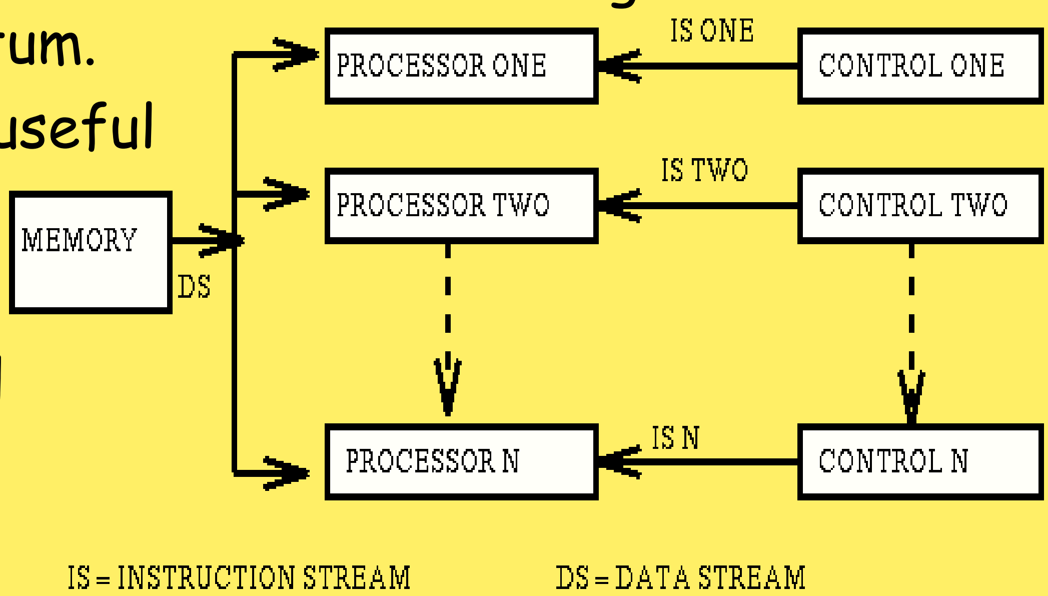
- Array processors such as the ICL DAP (Distributed Array Processor) and vector computers such as the CRAY 1 & 2 and CYBER 205 fit into the SIMD category.
- For Example, Adding two matrices A + B = C.
  Say we have two matrices A and B of order 2 and we have 4 processors.
- A11 + B11 = C11 ... A12 + B12 = C12
- A21 + B21 = C21 ... A22 + B22 = C22
- The same instruction is issued to all 4 processors (add the two numbers) and all processors execute the instructions simultaneously.
  It takes one step as opposed to four steps on a sequential machine.
- An instruction could be a simple one (eg adding two numbers) or a complex one (eg merging two lists of numbers). Similarly the datum may be simple (one number) or complex (several numbers).

- Sometimes it may be necessary to have only a subset of the processors execute an instruction i.e. only some data needs to be operated on for that instruction.

- This information can be encoded in the instruction itself indicating whether
  - the processor is **active** (execute the instruction)
  - the processor is **inactive** (wait for the next instruction)
- In most problems to be solved on SIMD (and MIMD) computers it is useful for the processors to be able to communicate with each other to exchange data or results. This can be done in two ways: by using a shared memory and shared variables or some form of interconnection network and message passing (distributed memory).

SDMI
Single Data Multiple Instruction (Pipelined processors)

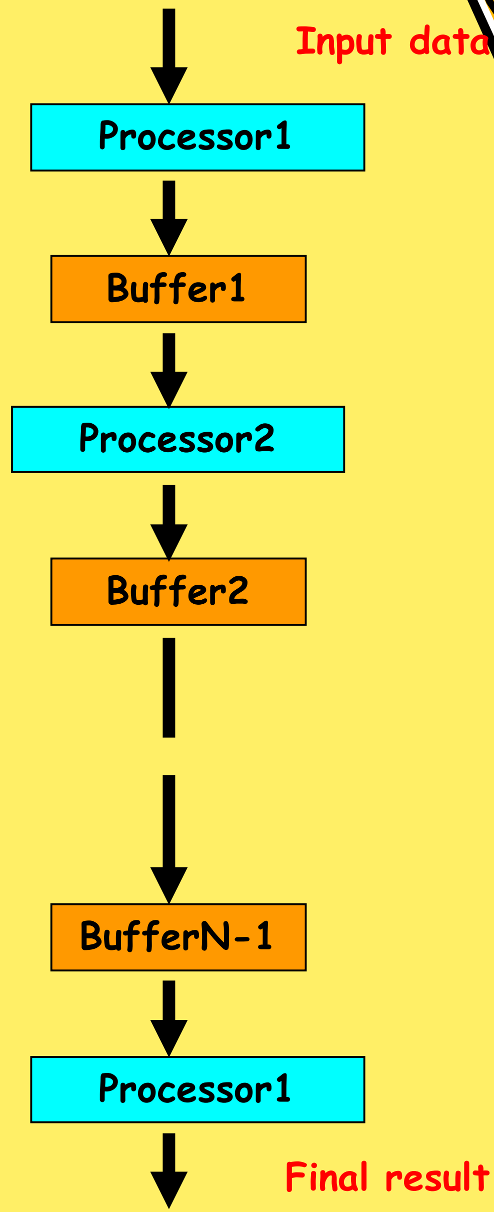
- N processors, each with its own control unit, share a common memory.
- There are N streams of instructions (algorithms / programs) and **one** stream of data. Parallelism is achieved by letting the processors do different things at the same time on the same datum.
- MISD machines are useful in computations where the same input is to be subjected to several different operations.



MISD
Multiple Instruction Single Data
(Pipelined processors)

- For example, Checking whether a number Z is prime. A simple solution is to try all possible divisions of Z. Assume the number of processors, N, is given by N = Z-2. All processors take Z as input and tries to divide it by its associated divisor. So it is possible in **one step** to check if Z is prime. More realistically if N < Z-2 then a subset of divisors would be assigned to each processor.
- For most applications MISD is no commercial machines exist with this design.
- However, a pipelined processors is another example of MISD.

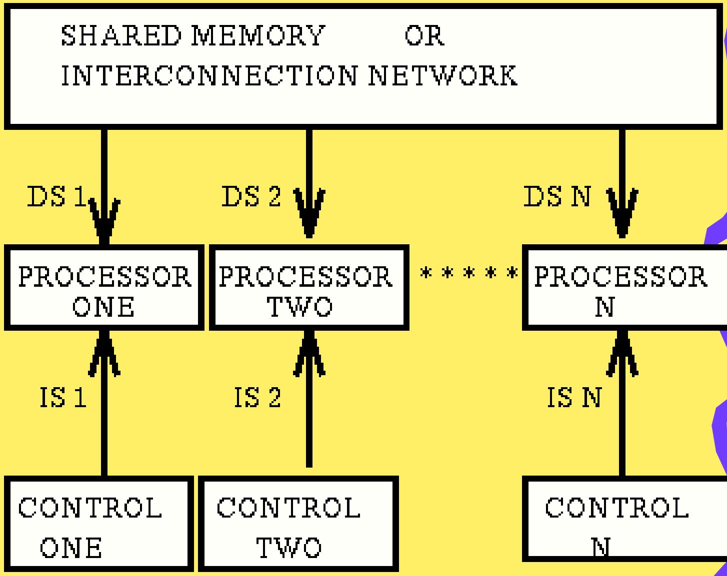
- Its basic idea is to break a complex time consuming function into a series of simpler sequential operations.
- The pipelined processors is then a machine that overlaps computations by subdividing and interleaving the operation of these sub-computations each on individual processor as happens in an assembly line fashion.
- The output of one stage is the input of the next.
- Buffers are used to hold output data temporally until the next stage finishes its data execution.
- Draw time overlap for different data streams



- Speedup = NM/N+M-1
- Speedup of a pipeline is defined to be: the number of times faster a pipeline component operates than a nonpipelined component that performs the same operation.
- N = no. of data streams
- M = Number of stages where each stage will require 1 time unit
- NM = total time taken to complete the N operations in nonpipelined component.
- For perfect usage of pipeline idea, time should be the same for all stages.
- If one stage takes less time, the remaining time will be wasted ???? How???

MIMD
Multiple **Instruction** Multiple **Data**
(Multiprocessors or multicomputers)

- This is the most general and most powerful of our classification. We have N processors, N streams of instructions and N streams of data.
- Each processor is fully programmable and can execute its own program.
- It can operate under the control of an instruction stream issued by its own control unit (i.e. each processor is capable of executing its own program on a different data.
- This means that the processors operate **asynchronously** (typically) i.e. can be doing different things on different data at the same time.



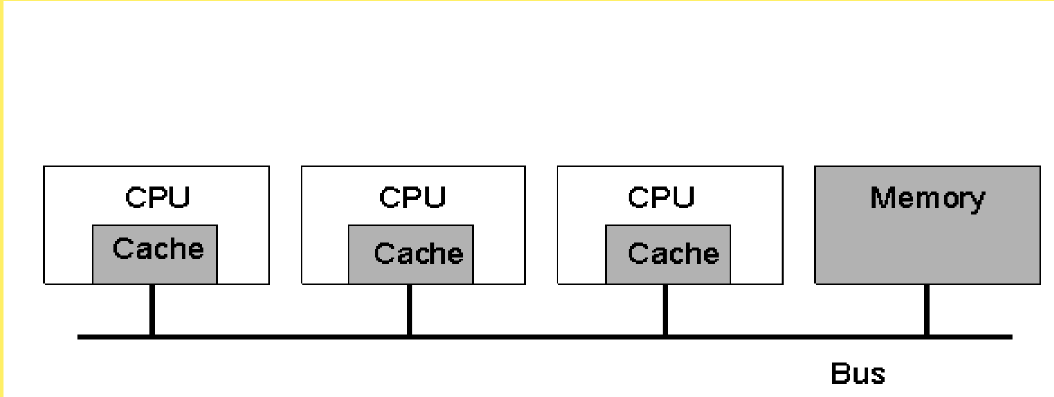
DS = DATA STREAM IS = INSTRUCTION STREAM

MIMD
Multiple **Instruction** Multiple **Data**
(Multiprocessors or multicomputers)

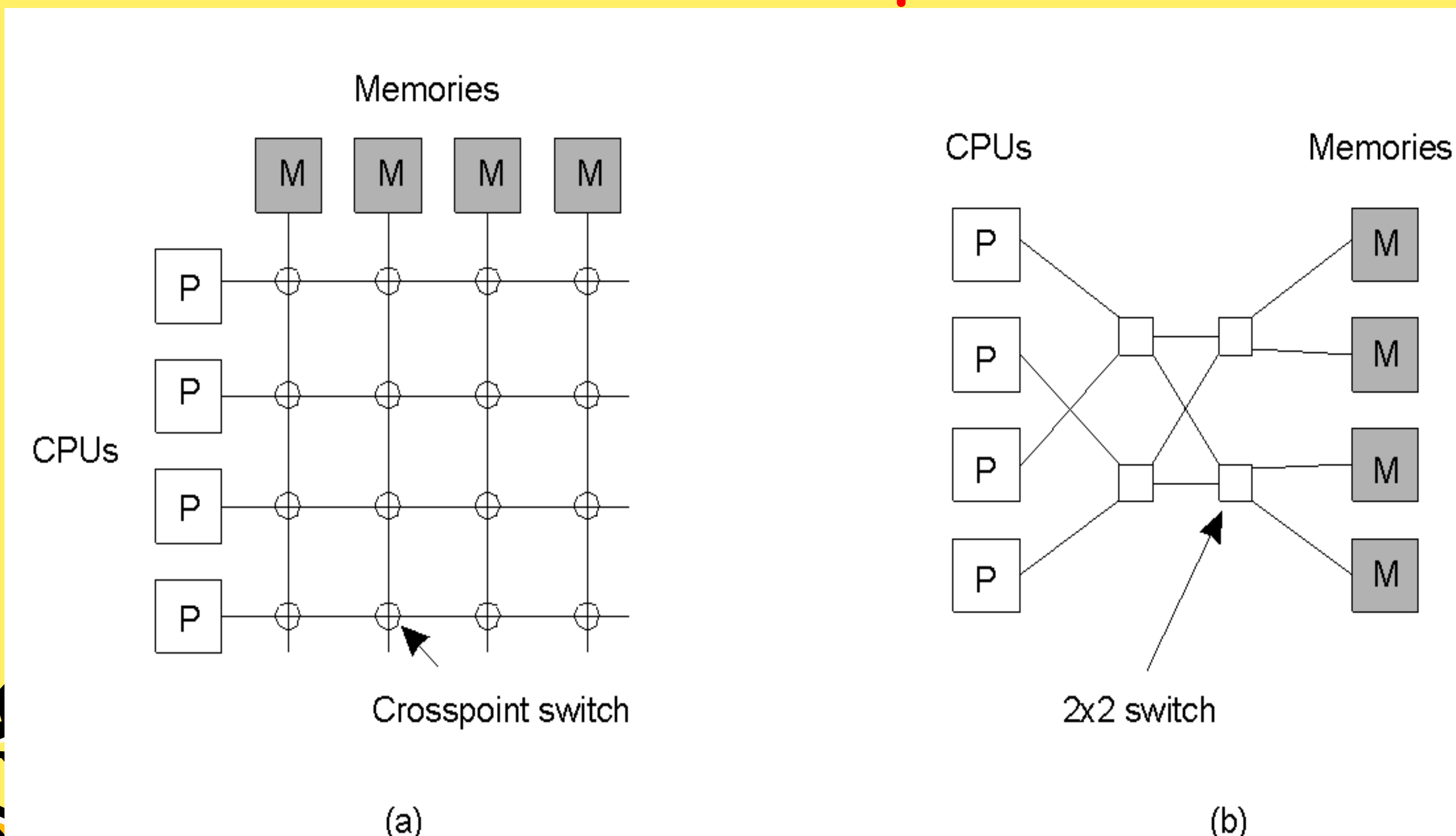
- As with SIMD computers, communication of data or results between processors can be via a shared memory or interconnection network.
- MIMD computers with shared memory are known as **multiprocessors** or **tightly coupled machines**. Examples are ENCORE, MULTIMAX, SEQUENT & BALANCE.
- MIMD computers with an interconnection network are known as **multicomputers** or **loosely coupled machines**. Examples are INTEL iPSC, NCUBE/7 and transputer networks.
- **<u>Note</u>**
  Multicomputers are sometimes referred to as distributed systems and sometimes not. Some text books in the literature claim that Distributed systems should work together to accomplish **one** job.

Tightly and loosely coupled systems

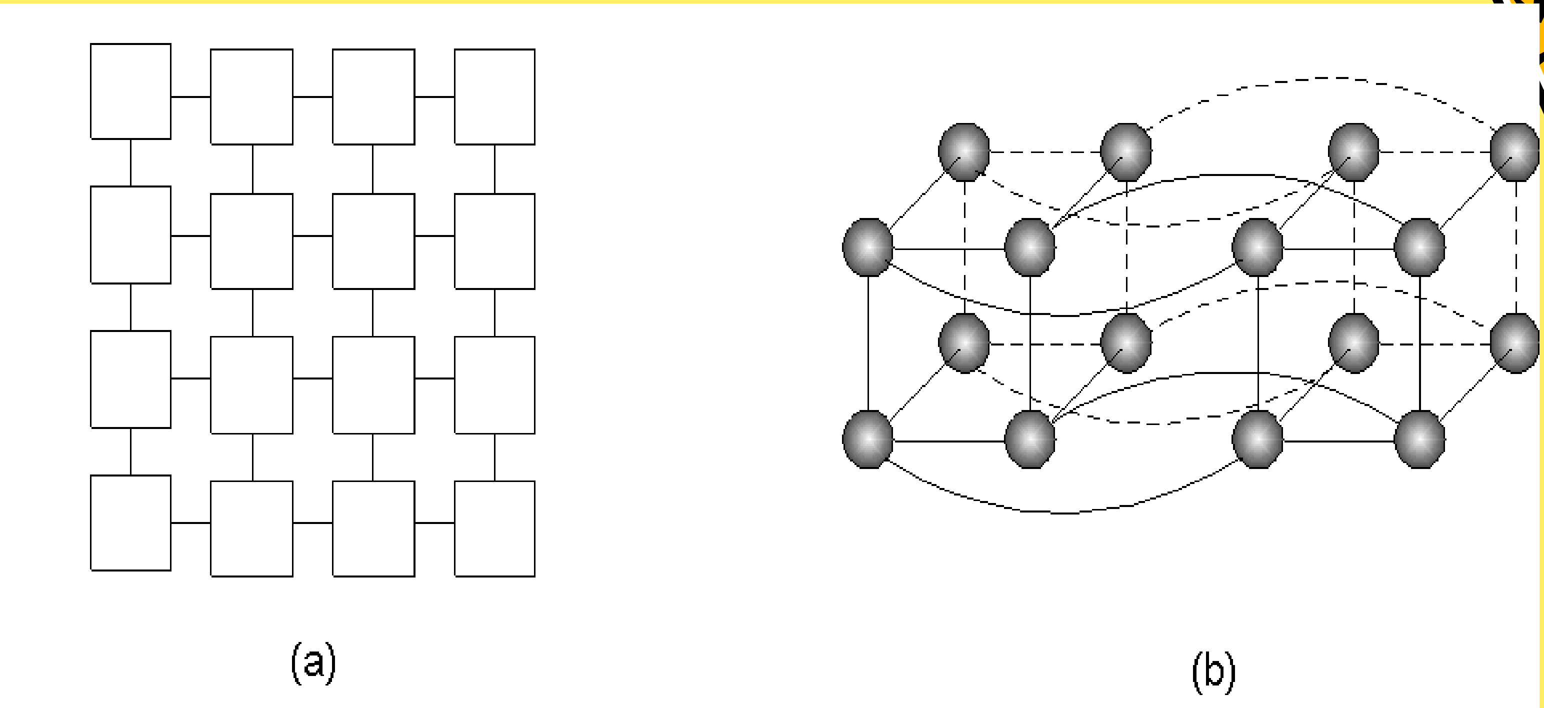
- Another dimension to our taxonomy is that in some systems the machines are tightly coupled and in others they are loosely coupled.
- <u>In a tightly-coupled system</u>, the delay experienced, when a message is sent from one computer to another is small, and the data rate is high that is, the number of bits per second that can be transferred is large.
- Typically, tightly coupled systems are referred to as multiprocessors when a number of processors each having its memory are organized on a single chassis and have access to common memory through high speed bus under control of single OS. Examples of this are: the <u>hypercube</u> and the <u>tree network</u> of processors.
- There must be strong interaction at HW and OS levels, at the dataset level, the program execution level.



A bus-based multiprocessor.



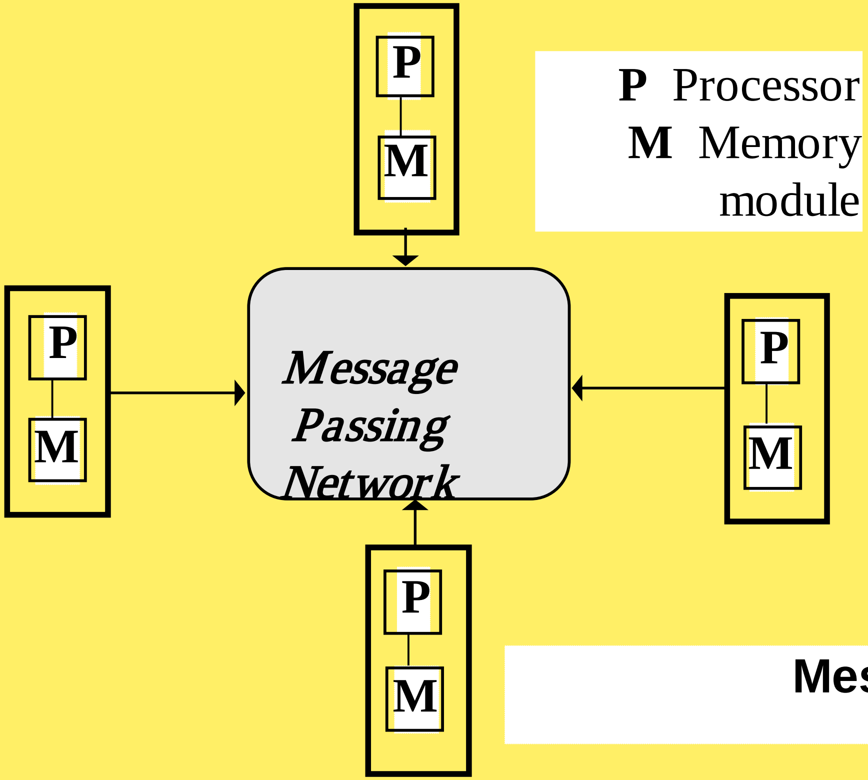
(a) A crossbar switch. (b) An omega switching network.



(a) Grid. (b) Hypercube multiprocessor.

- Grid
  - rectangular array of CPUs with connections only to neighbors
  - messages from one corner to the opposite corner in an n by n grid require (n-1)+(n-1) hops
- Hypercube
  - an n-dimensional cube with CPUs at the vertices
  - there are 2n CPUs
  - messages from one CPU to a CPU most distant from it require n hops (= log2(number of CPUs))

- In a loosely- coupled system, there are multiple computers (network) that communicate at the I/O level through intermachine message, thus the delay is large and the data rate is low.
- In order to complete a transfer of data between two processors, a cooperation is required between the two processors at the OS level.



Message Passing Architecture

- For example, two CPU chips on the same printed circuit board and connected by wires etched onto the board are likely to be tightly coupled, whereas two computers connected by a 2400 bit/sec modem over the telephone system are certain to be loosely coupled.
- Tightly-coupled systems tend to be used more as parallel systems (working on a single problem) and loosely-coupled ones tend to be used as distributed systems (working on many unrelated problems), although this is not always true.
- It is possible to have a system that is physically lossely coupled but logically tightly coupled. This could be made using OS constructs.
- Grid and clusters are examples of loosely coupled systems that can work in an orderly fashion to accomplish one job.
- When working to solve a certain problem, these processors should be organized using single operating system that organizes the interaction between them, assigns tasks to each processor and to ensure the synchronization of the assigned tasks.

Table [redacted] Computer Generations in 50 Years

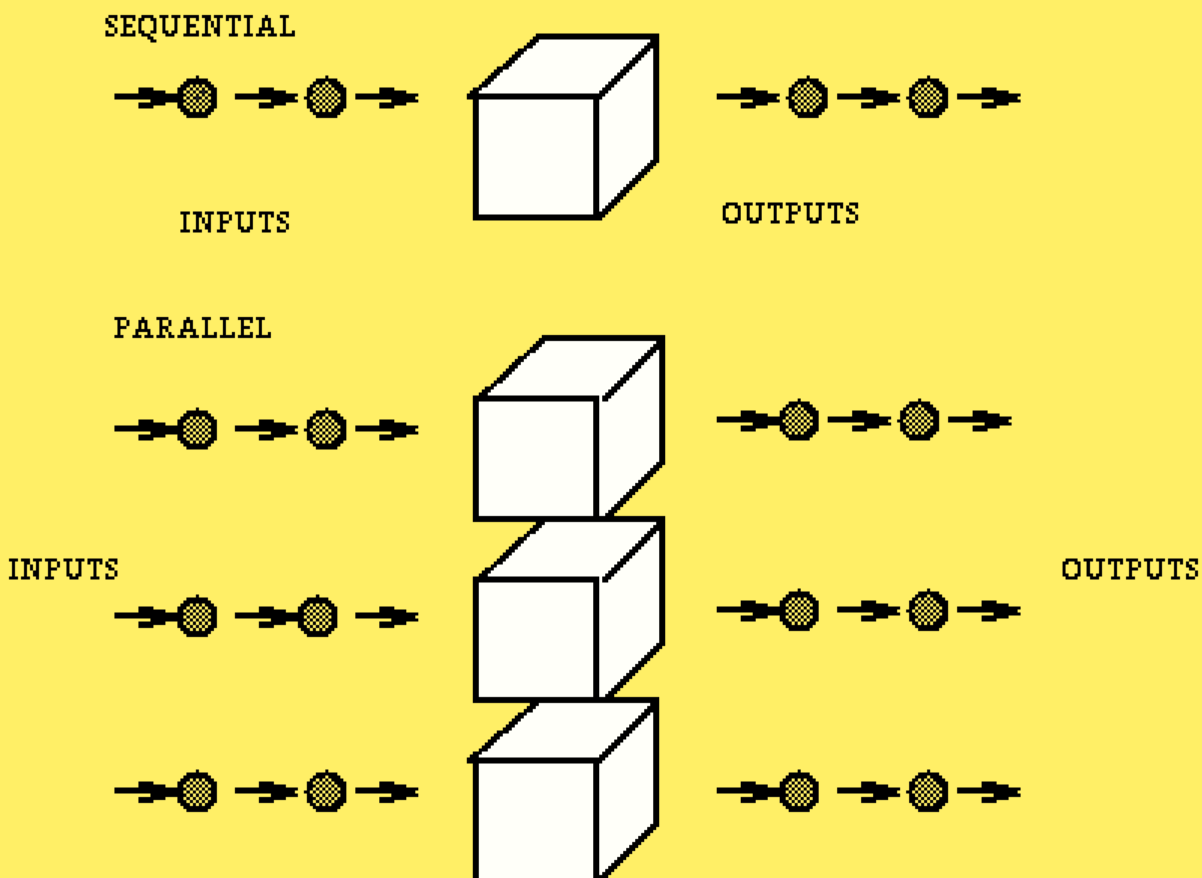
Generation Period	Technology and Architecture	Software and Operating System	Representative Systems
First **1946-1956**	Vacuum tubes and relay memory, single-bit CPU with accumulator-based instruction set	Machine/assembly languages, programs without subroutines	ENIAC, IBM 701, Princeton IAS
Second **1956-1967**	Discrete transistors, core memory, floating-point accelerator, I/O channels	Algol and Fortran with compilers, batch processing OS	IBM 7030, CDC 1604, Univac LARC
Third **1967-1978**	Integrated circuits, pipelined CPU, microprogrammed control unit	C language, multiprogramming, timesharing OS	PDP-11 IBM 360/370, CDC 6600
Fourth **1978-1989**	VLSI microprocessors, solid-state memory, multiprocessors, vector supercomputers	Symmetric multiprocessing, parallelizing compilers, message-passing libraries	IBM PC, VAX 9000, Cray X/MP
Fifth **1990- present**	ULSI circuits, scalable parallel computers, workstation clusters, Intranet, Internet	Java, microkernels, Multithreading, distributed OS, World-Wide Web	IBM SP2, SGI Origin 2000, Digital TruCluster

In hardware technology, the first generation used vacuum tubes and relay memory. The second generation was marked by the use of discrete transistors and core memory.

The Need For High Performance Computers

- Many of today's applications such as weather prediction, aerodynamics and artificial intelligence are very computationally intensive and require vast amounts of processing power.
- To calculate a 24 hour weather forecast for the many countries requires about 10^ 12 operations to be performed. This would take about 2.7 hours on a Cray-1 (capable of 10^ 8 operations per second).
- So to give accurate long range forecasts (e.g. a week) much more powerful computers are needed.
- One way of doing this is to use faster electronic components. The limiting factor is however the speed of light.
- The speed of light is 3 * 10^ 8 m/s.
- Considering two electronic devices (each capable of performing 10^ 12 operations per second) 0.5mm apart.
- It takes longer for a signal to travel between them than it takes for either of them to process it (10^ -12 seconds).
- So producing faster components is ultimately of no good.

- So it appears that the only way forward is to use PARALLELISM.
- The idea here is that if several operations can be performed simultaneously then the total computation time is reduced.
- The distributed system version has the **potential** of being 3 times as fast as the sequential machine.



Defining Speedup and Efficiency

- A **Parallel Algorithm** is an algorithm for the execution of a program which involves the running of two or more processes on two or more processors simultaneously.
- Two important measures of the quality of parallel algorithms are **speedup** and **efficiency**.
- If Ts is the time taken to run the fastest serial algorithm on one processor and if Tp is the time taken by a parallel algorithm on N processors then
- **Speedup** = SN = Ts / Tp

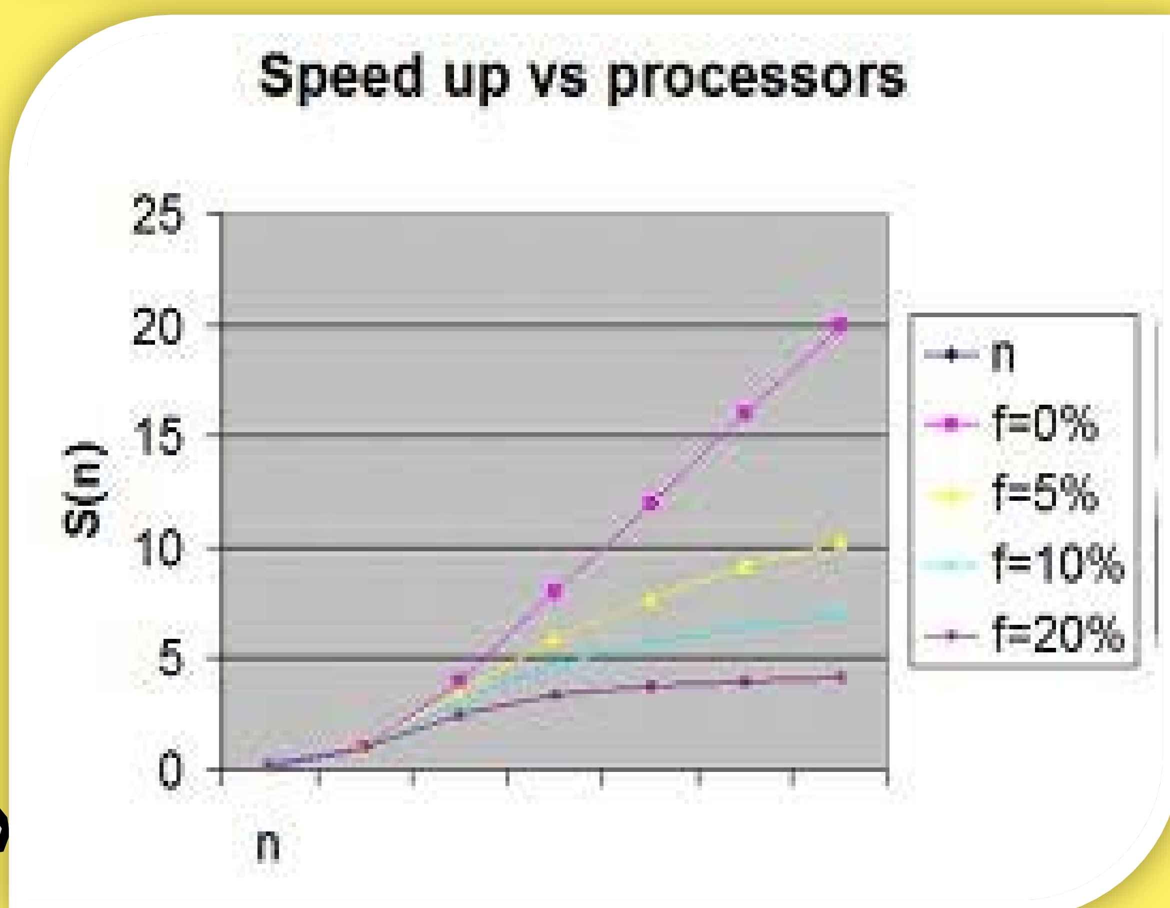
and the efficiency of the parallel algorithm is given by

- **Efficiency** = EN = SN / N

- If the best known serial algorithm takes 8 seconds i.e. Ts = 8, while a parallel algorithm takes 2 seconds using 5 processors, then
- SN = Ts / Tp = 8 / 2 = **4** and
- EN = SN / N = 4 / 5 = **0.8 = 80%**
- i.e. the parallel algorithm exhibits a speedup of 4 with 5 processors giving an 80% efficiency.

- A slightly <u>different definition</u> of speedup also exists.
- The time taken by the **parallel** algorithm on one processor divided by the time taken by the parallel algorithm on N processors.
- However, this is misleading since many parallel algorithms contain extra operations to accommodate the parallelism (e.g the communication) so the result is Ts is increased thus exaggerating the speedup.
- Which ever definition is used the ideal is to produce **linear speedup** i.e. produce a speedup of N using N processors and an efficiency of 1 (100%).
- However, in practice the speedup is reduced from its ideal value of N (the efficiency is bounded from above by 1).

Amdal's Law



Factors that limit speedup

1. **Software Overhead**

- Even with a completely equivalent algorithm, software overhead arises in the concurrent implementation. (e.g. there may be additional index calculations necessitated by the manner in which data are "split up" among processors.) i.e. there is generally more lines of code to be executed in the parallel program than the sequential program.

2. **Load Balancing**

- Speedup is generally limited by the speed of the slowest node. So an important consideration is to ensure that each node performs the same amount of work. i.e. the system is load balanced.

3. **Communication Overhead**

- Assuming that communication and calculation cannot be overlapped, then any time spent communicating the data between processors directly degrades the speedup. (because the processors are not calculating).

Distributed Systems Vs Centralized Systems

- The decision to build a distributed rather than a centralized system is often derived from the non-functional requirements and global nature of the applications.
- Scalability, openness, heterogeneity and fault tolerance are examples of such non-functional requirements.
- Distributed system can be defined as collection of autonomous hosts that are connected through a computer network.
- Each host executes components and operates a distributed middleware, which enables the components to coordinate their activities in such a way that users perceive the system as a single integrated computing facility.
- A host is a computer that executes components that form part of distributed system.
- A distributed system has more than one component on more than one host.

- Distributed systems usually employ some form of middleware that is layered between distributed system components and network operating system components

Advantages of Distributed Systems over Centralized Systems

Performance:

- very often a collection of processors can provide higher performance (and better price/performance ratio) than a centralized computer. A distributed system may have more computing power than a centralized mainframe or supercomputer

Distribution:

- many applications involve, by their nature, spatially separated machines (banking, commercial, automotive system).

Reliability (fault tolerance):

- if some of the machines crash, the system can survive.

Incremental growth:

- as requirements on processing power grow, new machines can be added incrementally.

Sharing of data/resources:

- shared data is essential to many applications (banking, computer supported cooperative work, reservation systems); other resources can be also shared (e.g. expensive printers).

Communication:

- facilitates human-to-human communication.

Disadvantages of Distributed Systems

Difficulties of developing distributed software:

- How should operating systems, programming languages and applications look like?

Networking problems:

- several problems are created by the network infrastructure, which have to be dealt with: loss of messages, overloading, ...

Security problems:

- sharing generates the problem of data security.

Characteristics of Distributed System

- In summary we can conclude that the careful design and implementation of a distributed system must consider some or all of the following characteristics.
- Resource sharing
  - Resource provided by a computer which is a member of a distributed system can be shared by clients and other members of the system via a network.
  - "Resource Manager" is a software module (based on set of management policies) that provides interfaces which enables resource to be manipulated by clients.
- Openness (extensible in various ways)
  - New resource sharing services can be incorporated without disruption or duplication of existing services.
  - every service is equally accessible to every client (local or remote);
  - it is easy to implement, install and debug new services;
  - users can write and install their own services.

- Transparency
  - Hide all unnecessary details from users.
  - Location transparency - clients do not need to know the location of the servers. Other types ????
- Share ability
  - Allows the comprising systems to use each other's resources.
- Expandability.
  - Permits new systems to be added as members of the overall system.
- Local Autonomy
  - Each processor is able to manage its local resources.
- Improved performance
  - Load balancing, resource replication.
  - Combined processing power of multiple computers provides much more processing Power than a centralized system with multiple CPUs.

- Several factors are influencing the performance of a
- distributed system:
  - The performance of individual workstations.
  - The speed of the communication infrastructure.
  - Extent to which reliability (fault tolerance) is provided (replication and preservation of coherence imply large overheads).
  - Flexibility in workload allocation: for example, idle processors (workstations) could be allocated automatically to a user's task.
- Improved reliability and availability
  - Disruption would not stop the whole system from providing its services as resources Spread across multiple computers.

- Concurrency
  - All concurrent access must be synchronized to avoid problems such as lost update, dirty read, incorrect summary & unrepeatable read.
- Scalability
  - Flexible to grow in size, efficiently utilize the new s/w & h/w added.
  - The system should remain efficient even with a significant increase in the number of users and resources connected:
  - cost of adding resources should be reasonable;
  - performance loss with increased number of users and resources should be controlled;
- Heterogeneity
  - Not having the same things like; programming languages, OSs, HW platforms, network protocols.
  - Distributed applications are typically heterogeneous:
  - different hardware: mainframes, workstations, PCs, servers, etc.;

- different software: UNIX,MSWindows, IBM OS/2, Real-time OSs, etc.;
- unconventional devices: teller machines, telephone switches, robots, manufacturing systems, etc.;
- diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

- The solution
  - *Middleware*, an additional software layer to mask heterogeneity
- Fault tolerance
  - The system appropriately handles errors occurred.
  - The system continues to work even if parts of it fails.
  - Accomplished via HW redundancy, software recovery.
  - **Fault-tolerance** is a main issue related to reliability: the system has to detect faults and act in a reasonable way:
  - *mask the fault*: continue to work with possibly reduced performance but without loss of data/ information.

- *fail gracefully*: react to the fault in a predictable way and possibly stop functionality for a short period, but without loss of data/information.

- Data on the system must not be lost, and copies stored redundantly on different servers must be kept **consistent**.
- The more copies kept, the better the availability, but keeping consistency becomes more difficult.

Transparency

- How to achieve the single system image?
- How to "fool" everyone into thinking that the collection of machines is a "simple" computer?
- Access transparency
- local and remote resources are accessed using identical operations.

Location transparency

- users cannot tell where hardware and software resources (CPUs, files, data bases) are located; the name of the resource shouldn't encode the location of the resource.

- Migration (mobility) transparency

- resources should be free to move from one location to another without having their names changed.

- Replication transparency

- the system is free to make additional copies of files and other resources (for purpose of performance and/or reliability), without the users noticing.

- Example: several copies of a file; at a certain request that copy is accessed which is the closest to the client.

- Concurrency transparency

- the users will not notice the existence of other users in the system (even if they access the same resources).

- Failure transparency

- applications should be able to complete their task despite failures occurring in certain components of the system.

- Performance transparency

- load variation should not lead to performance degradation.
- This could be achieved by automatic reconfiguration as response to changes of the load; it is difficult to achieve